Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 373 (2007) 163–181
Theoretical Computer Science
www.elsevier.com/locate/tcs

A rewriting logic framework for operational semantics of membrane systems

Oana Andrei[a], Gabriel Ciobanu[b,c,*], Dorel Lucanu[b]

[a] *LORIA, 54602 Villers-lès-Nancy Cedex, France*
[b] *Faculty of Computer Science, "A.I.Cuza" University, 700483 Iaşi, Romania*
[c] *Institute of Computer Science, Romanian Academy, 700505 Iaşi, Romania*

Abstract

Existing results in membrane computing refer mainly to P systems' characterization of Turing computability, also to some polynomial solutions to NP-complete problems by using an exponential workspace created in a "biological way". In this paper we define an operational semantics of a basic class of P systems, and give two implementations of the operational semantics using rewriting logic. We present some results regarding these implementations, including two operational correspondence results, and discuss why these implementations are relevant in order to take advantage of good features of both structural operational semantics and rewriting logic.

Keywords: Membrane systems; Operational semantics; Rewriting logic; Maude

1. Membrane systems

Membrane systems represent a new abstract model of parallel and distributed computing inspired by cell compartments and molecular membranes [16]. A cell is divided into various compartments, each compartment with a different task, with all of them working simultaneously to accomplish a more general task for the whole system. The membranes of a P system determine regions where objects and evolution rules can be placed. The objects evolve according to the rules associated with each region, and the regions cooperate in order to maintain the proper behaviour of the whole system. P systems provide a nice abstraction for parallel systems, and a suitable framework for distributed and parallel algorithms [9]. It is desirable to find more connections with various fields of computer science, including implementations and executable specifications. Sequential and parallel simulators exist, as well as a flexible web-based simulator available at http://psystems.ieat.ro.

A detailed description of P systems can be found in [17]. A *P system* consists of several membranes that do not intersect, and a *skin membrane*, surrounding them all. The membranes delimit *regions*, and contain multisets of *objects*, as well as *evolution rules*. The application of evolution rules is done in parallel, and is eventually

* Corresponding author at: Institute of Computer Science, Romanian Academy, 700505 Iaşi, Romania.
E-mail addresses: Oana.Andrei@loria.fr (O. Andrei), gabriel@iit.tuiasi.ro (G. Ciobanu), dlucanu@info.uaic.ro (D. Lucanu).

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2006.12.016

regulated by *priority* relationships between rules. The important feature of maximal parallelism is explained verbally in membrane systems literature. According to [17], maximal parallel rewriting "means that we assign objects to rules, non-deterministically choosing the objects and the rules, until no further assignment is possible. More mathematically stated, we look to the set of rules, and try to find a multiset of rules, by assigning multiplicities to rules, with two properties: (i) the multiset of rules is applicable to the multiset of objects available in the respective region, that is, there are enough objects in order to apply the rules a number of times as indicated by their multiplicities, and (ii) the multiset is maximal, no further rule can be added to it (because of the lack of available objects)".

The structure of a P system is represented by a tree (with the skin as its root), or equivalently, by a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. Graphically, a membrane structure is represented by a Venn diagram in which two sets can be either disjoint, or one the subset of the other. The membranes are labelled in a one-to-one manner. A membrane without any other membrane inside is said to be *elementary*.

Formally, a *P system* of degree m is a tuple $\Pi = (O, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_o)$, where:

(i) O is an alphabet of objects;
(ii) μ is a membrane structure;
(iii) w_i are the initial multisets over O associated with the regions defined by μ;
(iv) R_i are finite sets of evolution rules over O associated with the membranes, of typical form $u \to v$, with u a multiset over O and v a multiset containing paired symbols (messages) of the form $(c, here)$, (c, in_j), (c, out) and the dissolving symbol δ;
(v) ρ_i is a partial order relation over R_i, specifying a *priority* relation among the rules: $(r_1, r_2) \in \rho_i$ iff $r_1 > r_2$ (i.e., r_1 has a higher priority than r_2);
(vi) i_0 is either a number between 1 and m specifying the *output* membrane of Π, or it is equal to 0 indicating that the output is the outer region.

Since the skin is not allowed to be dissolved, we consider that the rules of the skin do not involve δ. These are called *general P systems*, or *transition P systems*; many other variants and classes have been introduced [17].

The membrane structure and the multisets in Π determine a configuration of the system. We can pass from a configuration to another one by using the evolution rules. The use of a rule $u \to v$ in a region with a multiset w means to subtract the multiset identified by u from w, and then to add the messages of v. The evolution rules in a membrane are applied in a maximal parallel manner, and all membranes evolves in parallel. Since the right-hand side v of a rule consists only of messages, an object introduced by a rule cannot evolve in the same step by means of another rule. If a message appears in v in the form $(c, here)$, then it remains in the same region. If it appears as (c, in_j), then a copy of c is introduced in the child membrane with the label j; if a child membrane with the label j does not exist, then the rule cannot be applied. If it appears as (c, out), then a copy of the object c is introduced in the surrounding membrane. If the special symbol δ appears in v, then the membrane which delimits the region is dissolved; in this way, all the objects in this region become elements of the surrounding membrane, while the rules of the dissolved membrane are removed.

Example 1. We consider an example of a P system with dissolving and priorities; it is taken from [17], page 71. This P system Π_1 generates values of the form n^2 for $n \geq 1$.

$$\Pi_1 = (O, \mu, w_1, w_2, w_3, (R_1, \rho_1), (R_2, \rho_2), (R_3, \rho_3), 1), \tag{1}$$

$$O = \{a, b, d, e, f\},\ \mu = [_1[_2[_3\ ]_3]_2]_1, \tag{2}$$

$$w_1 = \lambda, R_1 = \emptyset,\ \rho_1 = \emptyset,\ w_2 = \lambda, \rho_2 = \{(r_1, r_2)\}, \tag{3}$$

$$R_2 = \{b \to (d, here), d \to (de, here), r_1 : ff \to (f, here), r_2 : f \to \delta\}, \tag{4}$$

$$w_3 = af, R_3 = \{a \to (ab, here), a \to (b, here)\delta, f \to (ff, here)\}, \rho_3 = \emptyset. \tag{5}$$

The initial configuration is given in Fig. 1.

Since no object is available in membranes 1 and 2, the only possibility to start is by using the rules of membrane 3 together with its free objects a and f. Using the rules $a \to (ab, here)$ and $f \to (ff, here)$ in parallel for the available occurrences of a and f, after $n \geq 1$ steps we get n occurrences of b and 2^n occurrences of f. At any moment we can use $a \to (b, here)\delta$ instead of $a \to (ab, here)$, and consequently we get $n + 1$ occurrences of b and 2^{n+1} occurrences

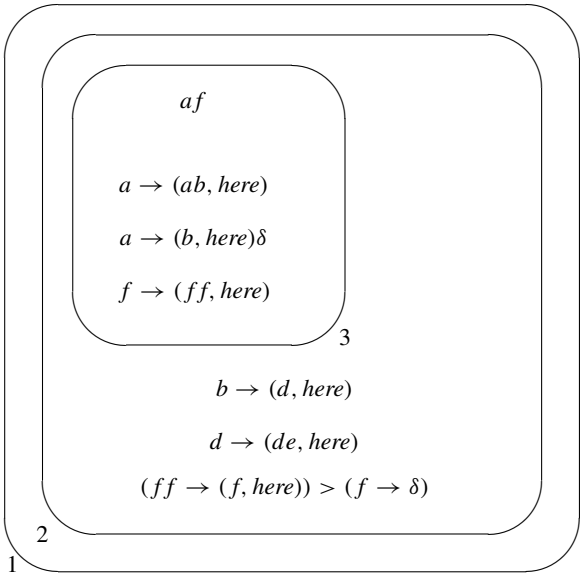

Fig. 1. The initial configuration of a P system generating n^2.

of f, followed by the process of dissolving membrane 3. Region 3 is dissolved, its rules are lost, and its objects move to region 2. The obtained configuration is $[_1\ [_2\ b^{n+1} f^{2^{n+1}},\ b \to (d, here),\ d \to (de, here),\ r_1 : ff \to (f, here),\ r_2 : f \to \delta,\ r_1 > r_2,\ ]_2\ ]_1$. According to the priority relation, the rule $ff \to f$ is used as much as possible. In one step b^{n+1} is transformed in d^{n+1}, while the number of f occurrences is divided by two. Then, in the next step, $n+1$ occurrences of e are produced, and the number of f occurrences is divided again by two. At each step, further $n+1$ occurrences of e are produced. Finally, after $n+1$ steps (n steps when the rule $ff \to (f, here)$ is used, and one when using the rule $f \to \delta$), membrane 2 is dissolved, its rules are removed, and its objects move to the skin region. The number of the objects e is the square of the number of d. Consequently, we may say that Π_2 generates values of the form n^2, for $n \geq 1$.

The existing results regarding P systems refer mainly to their expressive power and complexity, namely to their characterization of Turing computability (universality is obtained even with a small number of membranes, and with rather simple rules), and to polynomial solutions of NP-complete problems by using an exponential workspace created in a "biological way" (e.g. membrane division, string replication). Other types of formal results are given by normal forms, hierarchies and connections with various formalisms [16].

In this paper we refer to an operational semantics of P systems, discussing an encoding of P systems in rewriting logic, and proving soundness and correctness regarding this encoding. We show that rewriting logic is suitable for defining small-step operational semantics of P systems. Moreover, using rewriting logic, we were able to show that the big-step semantics of P systems can be refined leading to a richer class of models.

The structure of the paper is as follows. Section 2 presents the inductive definition of the membrane structure, the sets of the configurations for a P system, and an intuitive definition for the transition steps between the configurations. Section 3 presents an operational semantics of P systems considering each of the transition steps: maximal parallel rewriting, parallel communication, and parallel dissolving. Rewriting Logic and Maude are shortly described in Section 4, and then we implement the operational semantics of P systems using Maude. The relationship between the operational semantics of P systems and Maude rewriting is given in Section 5 by operational correspondence results. Conclusion, related work, and references end the paper.

2. Configurations and transitions

Let O be a finite alphabet of objects over which we consider the *free commutative monoid* O_c^*, whose elements are *multisets*. The empty multiset is denoted by *empty*.

Objects can be enclosed in messages together with a target indication. We have *here* messages of typical form $(w, here)$, *out* messages (w, out), and *in* messages (w, in_L). For the sake of simplicity, hereinafter we consider that the messages with the same target indication merge into one message:

$$\prod_{i\in I}(v_i, here) = (w, here), \prod_{i\in I}(v_i, in_L) = (w, in_L), \ \prod_{i\in I}(v_i, out) = (w, out),$$

with $w = \prod_{i\in I} v_i$, I a non-empty set, and $(v_i)_{i\in I}$ a family of multisets over O.

We use the mappings rules and $\mathsf{priority}$ to associate to a membrane label the set of evolution rules and the priority relation : $\mathsf{rules}(L_i) = R_i$, $\mathsf{priority}(L_i) = \rho_i$, and the projections L and w which return from a membrane its label and its current multiset.

The set $\mathcal{M}(\Pi)$ of membranes for a P system Π, and *the membrane structures* are inductively defined as follows:

- if L is a label, and w is a multiset over $O \cup (O \times \{here\}) \cup (O \times \{out\}) \cup \{\delta\}$, then $\langle\, L \mid w \,\rangle \in \mathcal{M}(\Pi)$; $\langle\, L \mid w \,\rangle$ is called *simple (or elementary) membrane*, and it has the structure $\langle\rangle$;
- if L is a label, w is a multiset over $O \cup (O \times \{here\}) \cup (O \times \{in_{L(M_j)} | j \in [n]\}) \cup (O \times \{out\}) \cup \{\delta\}$, $M_1, \ldots, M_n \in \mathcal{M}(\Pi), n \geq 1$, where each membrane M_i has the structure μ_i, then $\langle L \mid w;\ M_1, \ldots, M_n\rangle \in \mathcal{M}(\Pi)$; $\langle\, L \mid w\ ;\ M_1, \ldots, M_n \,\rangle$ is called *a composite membrane*, and it has the structure $\langle \mu_1, \ldots, \mu_n\rangle$.

We conventionally suppose the existence of a membrane *NULL* such that $M, NULL = M = NULL, M$ and $\langle\, L \mid w\ ;\ NULL \,\rangle = \langle\, L \mid w \,\rangle$. The use of *NULL* significantly simplifies several definitions and proofs. Let $\mathcal{M}^*(\Pi)$ be the free commutative monoid generated by $\mathcal{M}(\Pi)$ with the operation $(\_, \_)$ and the identity element *NULL*. We define $\mathcal{M}^+(\Pi)$ as the set of elements from $\mathcal{M}^*(\Pi)$ without the identity element. Let M_+, N_+ range over non-empty sets of sibling membranes, M_i over membranes, M_*, N_* range over possibly empty multisets of sibling membranes, and L over labels. The membranes preserve the initial labelling, evolution rules and priority relation among them in all subsequent configurations. Therefore in order to describe a membrane we consider its label and the current multiset of objects together with its structure.

A *configuration* for a P system Π is a skin membrane which has no messages and no dissolving symbol δ, i.e. the multisets of all regions are elements in O_c^*. We denote by $\mathcal{C}(\Pi)$ the set of configurations for Π.

An *intermediate configuration* is an arbitrary skin membrane in which we may find messages or the dissolving symbol δ. We denote by $\mathcal{C}^\#(\Pi)$ the set of intermediate configurations. We have $\mathcal{C}(\Pi) \subseteq \mathcal{C}^\#(\Pi)$.

Each P system has an initial configuration which is characterized by the initial multiset of objects for each membrane and the initial membrane structure of the system. For two configurations C_1 and C_2 of Π, we say that there is a *transition* from C_1 to C_2, and write $C_1 \Rightarrow C_2$, if the following *steps* are executed in the given order:

(1) *maximal parallel rewriting step*: each membrane evolves in a maximal parallel manner;
(2) *parallel communication of objects through membranes*, consisting in sending and receiving messages;
(3) *parallel membrane dissolving*, consisting in dissolving the membranes containing δ.

The last two steps take place only if there are messages or δ symbols resulting from the first step, respectively. If the first step is not possible, then neither are the other two steps; we say that the system has reached a *halting configuration*.

3. Operational semantics

We present shortly an operational semantics of P systems, considering each of the three steps.

3.1. Maximal parallel rewriting step

Here we formally define the maximal parallel rewriting $\overset{mpr}{\Longrightarrow}_L$ for a multiset of objects in one membrane, and we extend it to maximal parallel rewriting $\overset{mpr}{\Longrightarrow}$ over several membranes. Some preliminary notions are required.

Definition 1. The irreducibility property w.r.t. the maximal parallel rewriting relation for multisets of objects, membranes, and for sets of sibling membranes is defined as follows:

- a multiset of messages and the dissolving symbol δ are **L-irreducible**;

- a multiset of objects w is L**-irreducible** iff there are no rules in $\mathsf{rules}(L)$ applicable to w with respect to the priority relation $\mathsf{priority}(L)$;
- a simple membrane $\langle\ L \mid w\ \rangle$ is **mpr-irreducible** iff w is L-irreducible;
- a nonempty set of sibling membranes $M_1, \ldots, M_n$ is **mpr-irreducible** iff M_i is mpr-irreducible for every $i \in [n]$; *NULL* is **mpr-irreducible**;
- a composite membrane $\langle\ L \mid w\ ;\ M_1, \ldots, M_n\ \rangle$ is **mpr-irreducible** iff w is L-irreducible, and the set of sibling membranes $M_1, \ldots, M_n$ is mpr-irreducible.

The priority relation is a form of control on the application of rules. In the presence of a priority relation, no rule of a lower priority can be used during the same evolution step when a rule with a higher priority is used, even if the two rules do not compete for the same objects. We formalize the conditions imposed by the priority relation on rule applications in the definition below.

Definition 2. Let M be a membrane labelled by L, and w a multiset of objects. A non-empty multiset $R = (u_1 \to v_1, \ldots, u_n \to v_n)$ of evolution rules is (L, w)**-consistent** if:

- $R \subseteq \mathsf{rules}(L)$,
- $w = u_1 \ldots u_n z$, so each rule $r \in R$ is applicable on w,
- $(\forall r \in R, \forall r' \in \mathsf{rules}(L))$ r' applicable on w implies $(r', r) \notin \mathsf{priority}(L)$,[1]
- $(\forall r', r'' \in R)$ $(r', r'') \notin \mathsf{priority}(L)$,
- the dissolving symbol δ has at most one occurrence in the multiset $v_1 \ldots v_n$.

Maximal parallel rewriting relations $\overset{mpr}{\Longrightarrow}_L$ and $\overset{mpr}{\Longrightarrow}$ are defined by the following inference rules:

For each $w = u_1 \ldots u_n z \in O_c^+$ such that z is L-irreducible, and (L, w)-consistent rules $(u_1 \to v_1, \ldots, u_n \to v_n)$,

$$(\mathbf{R_1})\ \frac{}{u_1 \ldots u_n z \overset{mpr}{\Longrightarrow}_L v_1 \ldots v_n z}.$$

For each $w \in O_c^+$, $w' \in (O \cup Msg(O) \cup \{\delta\})_c^+$, and mpr-irreducible $M_* \in \mathcal{M}^*(\Pi)$,

$$(\mathbf{R_2})\ \frac{w \overset{mpr}{\Longrightarrow}_L w'}{\langle\ L \mid w\ ;\ M_*\ \rangle \overset{mpr}{\Longrightarrow} \langle\ L \mid w'\ ;\ M_*\ \rangle}.$$

For each L-irreducible $w \in O_c^*$, and $M_+, M'_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_3})\ \frac{M_+ \overset{mpr}{\Longrightarrow} M'_+}{\langle\ L \mid w\ ;\ M_+\ \rangle \overset{mpr}{\Longrightarrow} \langle\ L \mid w\ ;\ M'_+\ \rangle}.$$

For each $w \in O_c^+$, $w' \in (O \cup Msg(O) \cup \{\delta\})_c^+$, $M_+, M'_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_4})\ \frac{w \overset{mpr}{\Longrightarrow}_L w',\ M_+ \overset{mpr}{\Longrightarrow} M'_+}{\langle\ L \mid w\ ;\ M_+\ \rangle \overset{mpr}{\Longrightarrow} \langle\ L \mid w'\ ;\ M'_+\ \rangle}.$$

For each $M, M' \in \mathcal{M}(\Pi)$, and $M_+, M'_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_5})\ \frac{M \overset{mpr}{\Longrightarrow} M',\ M_+ \overset{mpr}{\Longrightarrow} M'_+}{M, M_+ \overset{mpr}{\Longrightarrow} M', M'_+}.$$

For each $M, M' \in \mathcal{M}(\Pi)$, and mpr-irreducible $M_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_6})\ \frac{M \overset{mpr}{\Longrightarrow} M'}{M, M_+ \overset{mpr}{\Longrightarrow} M', M_+}.$$

[1] We recall that $(r_1, r_2) \in \mathsf{priority}(L)$ iff $r_1 > r_2$.

We note that $\stackrel{mpr}{\Longrightarrow}$ for simple membranes can be described by rule $(\mathbf{R_2})$ with $M_* = NULL$.

Remark 1. The relations $\stackrel{mpr}{\Longrightarrow}$ and mpr-irreducibility are connected as expected: M is mpr-irreducible iff there does not exist M' such that $M \stackrel{mpr}{\Longrightarrow} M'$.

Proposition 1. *Let Π be a P system. If $C \in \mathcal{C}(\Pi)$ and $C' \in \mathcal{C}^{\#}(\Pi)$ such that $C \stackrel{mpr}{\Longrightarrow} C'$, then C' is mpr-irreducible.*

The proof of Proposition 1 follows by structural induction on C.

The formal definition of $\stackrel{mpr}{\Longrightarrow}$ given above corresponds to the intuitive description of the maximal parallelism, namely the application of a multiset of rules such that no further rule can be added to these rules. The nondeterminism is given by the associativity and commutativity of the concatenation operation over objects used in $(\mathbf{R_1})$. The parallelism of the evolution rules in a membrane is also given by $(\mathbf{R_1})$: $u_1 \dots u_n z \stackrel{mpr}{\Longrightarrow}_L v_1 \dots v_n z$ says that the rules of the multiset $(u_1 \to v_1, \dots, u_n \to v_n)$ are applied simultaneously. The fact that the membranes are evolving in parallel is described by $(\mathbf{R_3})$–$(\mathbf{R_6})$.

3.2. Parallel communication of objects

We say that a multiset w is *here-free/out-free/in$_L$-free* if it does not contain any *here/out/in$_L$* messages, respectively. For w a multiset of objects and messages, we introduce the operations obj, here, out, and in_L as follows:

$\mathsf{obj}(w)$ is obtained from w by removing all messages,

$$\mathsf{here}(w) = \begin{cases} empty & \text{if } w \text{ is } here\text{-free,} \\ w'' & \text{if } w = w'(w'', here) \wedge w' \text{ is } here\text{-free;} \end{cases}$$

$$\mathsf{out}(w) = \begin{cases} empty & \text{if } w \text{ is } out\text{-free,} \\ w'' & \text{if } w = w'(w'', out) \wedge w' \text{ is } out\text{-free;} \end{cases}$$

$$\mathsf{in}_L(w) = \begin{cases} empty & \text{if } w \text{ is } in_L\text{-free,} \\ w'' & \text{if } w = w'(w'', in_L) \wedge w' \text{ is } in_L\text{-free.} \end{cases}$$

We consider the extension of the operator w (previously defined over membranes) to nonempty sets of sibling membranes by setting $\mathsf{w}(NULL) = empty$ and $\mathsf{w}(M_1, \dots, M_n) = \mathsf{w}(M_1) \dots \mathsf{w}(M_n)$.

We recall that the messages with the same target merge in one message.

Definition 3. The **tar-irreducibility** property for membranes and for sets of sibling membranes is defined as follows:

- a simple membrane $\langle\, L \mid w \,\rangle$ is **tar-irreducible** iff w is *here*-free and $L \neq Skin \vee (L{=}Skin \wedge w$ *out*-free);
- a nonempty set of sibling membranes $M_1, \dots, M_n$ is **tar-irreducible** iff M_i is tar-irreducible for every $i \in [n]$; *NULL* is **tar-irreducible**;
- a composite membrane $\langle\, L \mid w\ ;\ M_1, \dots, M_n \,\rangle$, $n \geq 1$, is **tar-irreducible** iff: w is *here*-free and *in$_{L(M_i)}$*-free for every $i \in [n]$, $L \neq Skin \vee (L = Skin \wedge w$ is *out*-free), $\mathsf{w}(M_i)$ is *out*-free for all $i \in [n]$, and the set of sibling membranes $M_1, \dots, M_n$ is tar-irreducible;

Notation. We treat messages of the form $(w', here)$ as a particular communication inside a membrane, and we substitute $(w', here)$ by w'. We denote by $\overline{w}$ the multiset obtained by replacing $(\mathsf{here}(w), here)$ with $\mathsf{here}(w)$ in w. For instance, if $w = a\,(bc, here)\,(d, out)$ then $\overline{w} = abc\,(d, out)$, where $\mathsf{here}(w) = bc$. We note that $\mathsf{in}_L(\overline{w}) = \mathsf{in}_L(w)$, and $\mathsf{out}(\overline{w}) = \mathsf{out}(w)$.

Parallel communication relation $\stackrel{tar}{\Longrightarrow}$ is defined by the following inference rules:

For each tar-irreducible $M_* \in \mathcal{M}^*(\Pi)$ and multiset w such that
$\mathsf{here}(w) \neq empty$, or $L = Skin \wedge \mathsf{out}(w) \neq empty$, or it exists $M_i \in M_*$ with
$\mathsf{in}_{L(M_i)}(w)\mathsf{out}(\mathsf{w}(M_i)) \neq empty$,

$$(\mathbf{C_1})\ \frac{}{\langle\, L \mid w\ ;\ M_* \,\rangle \stackrel{tar}{\Longrightarrow} \langle\, L \mid w'\ ;\ M'_* \,\rangle}$$

where

$$w' = \begin{cases} \mathsf{obj}(\overline{w})\,\mathsf{out}(\mathsf{w}(M_*)) & \text{if } L = \mathit{Skin}, \\ \mathsf{obj}(\overline{w})\,(\mathsf{out}(w), \mathit{out})\,\mathsf{out}(\mathsf{w}(M_*)) & \text{otherwise}, \end{cases}$$

and

$$\mathsf{w}(M_i') = \mathsf{obj}(\mathsf{w}(M_i'))\,\mathsf{in}_{\mathsf{L}(M_i)}(w), \text{ for all } M_i \in M_*.$$

For each $M_1, \ldots, M_n, M_1', \ldots, M_n' \in \mathcal{M}^+(\Pi)$, and multiset w,

$$(\mathbf{C_2})\ \frac{M_1, \ldots, M_n \overset{tar}{\Longrightarrow} M_1', \ldots, M_n'}{\langle\, L \mid w\ ;\ M_1, \ldots, M_n \,\rangle \overset{tar}{\Longrightarrow} \langle\, L \mid w''\ ;\ M_1'', \ldots, M_n'' \,\rangle}$$

where

$$w'' = \begin{cases} \mathsf{obj}(\overline{w})\,\mathsf{out}(\mathsf{w}(M_1', \ldots, M_n')) & \text{if } L = \mathit{Skin}, \\ \mathsf{obj}(\overline{w})\,(\mathsf{out}(w), \mathit{out})\,\mathsf{out}(\mathsf{w}(M_1', \ldots, M_n')) & \text{otherwise}, \end{cases}$$

and each M_i'' is obtained from M_i' by replacing its resources with

$$\mathsf{w}(M_i'') = \mathsf{obj}(\overline{\mathsf{w}(M_i')})\,\mathsf{in}_{\mathsf{L}(M_i')}(w), \text{ for all } i \in [n].$$

For each $M, M' \in \mathcal{M}(\Pi)$, and tar-irreducible $M_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{C_3})\ \frac{M \overset{tar}{\Longrightarrow} M'}{M, M_+ \overset{tar}{\Longrightarrow} M', M_+}.$$

For each $M \in \mathcal{M}(\Pi)$, $M_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{C_4})\ \frac{M \overset{tar}{\Longrightarrow} M',\ M_+ \overset{tar}{\Longrightarrow} M_+'}{M, M_+ \overset{tar}{\Longrightarrow} M', M_+'}.$$

Remark 2. M is tar-irreducible iff there does not exist M' such that $M \overset{tar}{\Longrightarrow} M'$.

Proposition 2. *Let Π be a P system. If $C \in \mathcal{C}^{\#}(\Pi)$ with messages and $C \overset{tar}{\Longrightarrow} C'$, then C' is tar-irreducible.*

The proof of Proposition 2 is by structural induction on C.

3.3. Parallel membrane dissolving

If the special symbol δ occurs in the multiset of objects of a membrane labelled by L, that membrane is dissolved, its evolution rules and the associated priority relation are lost, and its contents (objects and membranes) is added to the contents of the surrounding membrane. We say that a multiset w is δ-*free* if it does not contain the special symbol δ.

Definition 4. The δ-**irreducibility** property for membranes and for sets of sibling membranes is defined as follows:

- a simple membrane is δ-**irreducible** iff it has no messages;
- a non-empty set of sibling membranes $M_1, \ldots, M_n$ is δ-**irreducible** iff every membrane M_i is δ-irreducible, for $1 \leq i \leq n$;
- a composite membrane $\langle\, L \mid w\ ;\ M_+ \,\rangle$ is δ-**irreducible** iff w has no messages, M_+ is δ-irreducible, and $w(M_+)$ is δ-free;
- *NULL* is δ-**irreducible**.

Parallel dissolving relation $\overset{\delta}{\Longrightarrow}$ is defined by the following inference rules:

For each $M_* \in \mathcal{M}^*(\Pi)$, δ-irreducible $\langle\, L_2 \mid w_2\delta\ ;\ M_* \,\rangle$, and label L_1,

$$(\mathbf{D_1})\ \frac{}{\langle\, L_1 \mid w_1\ ;\ \langle\, L_2 \mid w_2\delta\ ;\ M_* \,\rangle \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1 w_2\ ;\ M_* \,\rangle}.$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, $M'_* \in \mathcal{M}^*(\Pi)$, δ-free multiset w_2, multisets w_1, w'_2, and labels L_1, L_2

$$(\mathbf{D_2})\ \frac{\langle\, L_2 \mid w_2\ ;\ M_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L_2 \mid w'_2\ ;\ M'_* \,\rangle}{\langle\, L_1 \mid w_1\ ;\ \langle\, L_2 \mid w_2\ ;\ M_+ \,\rangle \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1\ ;\ \langle\, L_2 \mid w'_2\ ;\ M'_* \,\rangle \,\rangle}.$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, $M'_* \in \mathcal{M}^*(\Pi)$, multisets w_1, w_2, w'_2, and labels L_1, L_2

$$(\mathbf{D_3})\ \frac{\langle\, L_2 \mid w_2\delta\ ;\ M_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L_2 \mid w'_2\delta\ ;\ M'_* \,\rangle}{\langle\, L_1 \mid w_1\ ;\ \langle\, L_2 \mid w_2\delta\ ;\ M_+ \,\rangle \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1 w'_2\ ;\ M'_* \,\rangle}.$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, $M'_*, N'_* \in \mathcal{M}^*(\Pi)$, δ-irreducible $\langle\, L \mid w\ ;\ N_+ \,\rangle$, and multisets w', w'',

$$(\mathbf{D_4})\ \frac{\langle\, L \mid w\ ;\ M_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L \mid w'\ ;\ M'_* \,\rangle}{\langle\, L \mid w\ ;\ M_+, N_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L \mid w'\ ;\ M'_*, N_+ \,\rangle}.$$

$$(\mathbf{D_5})\ \frac{\langle\, L \mid w\ ;\ M_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L \mid ww'\ ;\ M'_* \,\rangle \quad \langle\, L \mid w\ ;\ N_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L \mid ww''\ ;\ N'_* \,\rangle}{\langle\, L \mid w\ ;\ M_+, N_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L \mid ww'w''\ ;\ M'_*, N'_* \,\rangle}.$$

Remark 3. M is δ-irreducible iff there does not exist M' such that $M \overset{\delta}{\Longrightarrow} M'$.

Proposition 3. *Let Π be a P system. If $C \in \mathcal{C}^{\#}(\Pi)$ is tar-irreducible and $C \overset{\delta}{\Longrightarrow} C'$, then C' is δ-irreducible.*

The proof of Proposition 3 follows by a structural induction on C.

We can note that $C \in \mathcal{C}(\Pi)$ iff C is tar-irreducible and δ-irreducible.

According to the standard description in membrane computing, a *transition step* between two configurations $C, C' \in \mathcal{C}(\Pi)$ is given by: $C \Rightarrow C'$ iff C and C' are related by one of the following relations:

$$\text{either } C \overset{mpr}{\Longrightarrow}; \overset{tar}{\Longrightarrow} C',\ \text{or}\ C \overset{mpr}{\Longrightarrow}; \overset{\delta}{\Longrightarrow} C',\ \text{or}\ C \overset{mpr}{\Longrightarrow}; \overset{tar}{\Longrightarrow}; \overset{\delta}{\Longrightarrow} C'.$$

The three alternatives in defining $C \Rightarrow C'$ are given by the existence of messages and dissolving symbols along the system evolution. Starting from a configuration without messages and dissolving symbols, we apply the "mpr" rules and get an intermediate configuration which is mpr-irreducible; if we have messages, then we apply the "tar" rules and get an intermediate configuration which is tar-irreducible; if we have dissolving symbols, then we apply the dissolving rules and get a configuration which is δ-irreducible. If the last configuration has no messages or dissolving symbols, then we say that the transition relation $\Rightarrow$ is well-defined as an evolution step between the first and last configurations.

Proposition 4. *The relation $\Rightarrow$ is well-defined over the entire set $\mathcal{C}(\Pi)$ of configurations.*

Examples of inference trees, as well as the proofs of the results are presented in [2]. We have shortly presented the operational semantics just to give sense to the implementations of P systems into rewriting logic.

4. Implementing P systems using rewriting logic

A general methodology for using rewriting logic to give semantics to programming languages is presented in [15]. The authors describe the advantages and the drawbacks of equational specifications and structural operational semantics [19], as well as the capabilities of rewriting logic to unify the two approaches such that their advantages are preserved and their drawbacks are surmounted. By using an efficient implementation of rewriting logic as Maude [11], a formal specification of a language can be automatically transformed into an interpreter. Moreover, Maude provides an useful search command, a semi-decision procedure for finding failures of safety properties, and also a model checker. Roughly speaking, a rewrite theory is a triple $(\Sigma, E, \mathcal{R})$, where (Σ, E) is an equational theory used for implementing the deterministic computation, therefore (Σ, E) should be terminating and Church-Rosser, and $\mathcal{R}$ is a set of rewrite rules used to implement nondeterministic and/or concurrent computations. Therefore we find rewriting

logic suitable for implementing the membrane systems. The Web page of the implementations described in this paper can be found at http://thor.info.uaic.ro/~rewps/index.html.

We mainly follow the same line as that in [15]: we use an equational part for specifying configurations (configurations are similar to states), and rewrite rules for specifying the semantics. In [15], a continuation passing style is used for defining the dynamics of a program. This approach is suitable for languages with complex control structures. In our case, the control part of a P system is different from that of a programming language. It consists in maximal parallel application of the evolution rules according to their priorities (if any), and the (repeated) steps of internal evolution, communication, and dissolving. This sequence of steps uses a kind of synchronization; therefore we cannot follow the continuation passing style.

The first challenge is to describe the maximal parallel rewriting, because this is not quite natural for rewriting logic. In [1] the reflection property of rewriting logic is used for defining maximal parallel rewriting. A P system was defined at object level, and its semantics at metalevel. The description at metalevel assumes many additional operations, and therefore the checking and the analysis of the result specification was time consuming. Here we use a novel approach. The evolution rules are represented as terms at the object level. This allows us to define the operational semantics at the object level and, consequently, the checking and the analysis of the result specification is more efficient.

The second challenge is given by the sequence of internal steps: evolution, communication, and dissolving. We decorated the terms denoting membranes with colours, and these colours are used by a rewrite engine to choose the appropriate (sub)set of rewriting rules. The definition of semantics for P systems in rewriting logic reveals an interesting aspect: internal evolution, communication and dissolving inside a complex membrane may interleave. This leads to a relaxed operational semantics for P systems presented in Section 5.1. If only main configurations are observable, then the big-step semantics and the refined one are behaviourally equivalent.

4.1. Maude equational specification of P systems

We consider the sorts `Obj`, `Soup`, `Membrane`, `Configuration` for object names, multisets of ingredients, membranes and configurations, respectively. Their definitions and the relations between them are introduced by the following (simplified) functional module:

```
mod PSCONFIGURATION is
 pr QID .
 sort Label . subsort Label < Qid .
 sorts Obj Soup EmptyMembraneSet Membrane MembraneSet Configuration .
 subsort Obj < Soup .
 subsort EmptyMembraneSet Membrane < MembraneSet .
 op empty : -> Soup .
 op __ : Soup Soup -> Soup [assoc comm id: empty] .
 op NULL : -> EmptyMembraneSet .
 op <_|_> : Label Soup -> Membrane .
 op _‘,_ : MembraneSet MembraneSet -> MembraneSet
           [assoc comm id: NULL] .
 op <_|_;_> : Label Soup MembraneSet -> Membrane .
 op ‘{_‘} : Membrane -> Configuration .
endm
```

The sort `Soup` represents the multiset type with two constructors: `empty` and `_ _`. The second constructor is required to satisfy the structural laws of associativity and commutativity, and it has an identity `empty`. The subsort relation `Obj < Soup` says that each object defines a particular multiset. The sort `Membrane` has two constructors for the two types of membranes: elementary and composite. The operation `_ , _` on a set of membranes is required to satisfy the structural laws of associativity, commutativity and identity because the order of sibling membranes is irrelevant; what matters is the positions of the membranes, namely if one is inside the other. An expression of the form $\langle L \mid W \rangle$ corresponds to a state of an elementary membrane labelled by L and having the multiset of objects given by W, while an expression of the form $\langle L \mid W \ ; \ M_1, \ldots, M_n \rangle$ corresponds to a composite membrane labelled by L with the current multiset of objects described by W and the state of the i-th component given by M_i. The sort `Configuration` corresponds to a P

system configuration; it has no subsorts and its constructor $\{\_\} : \mathit{Membrane} \rightarrow \mathit{Configuration}$ has the skin membrane as its only argument. The intended meaning consists in starting our maximal parallel rewriting strategy only from the skin membrane in order to ensure the processing of all membranes of P system during a transition step.

For the communications of objects through membranes, we define a sort `Target` together with two operations for sending objects *out* of the membrane or *in* a specified membrane, such that a pair composed of a multiset of objects and a target represents an element of sort `Message`.

In order to deal with the membrane dissolving, we add a sort `Dissolve`, and a constant operation `delta` corresponding to the dissolving element δ.

For the evolution rules we introduce a sort `MRule` together with two operations corresponding to rules with or without priorities:

```
op _:_->_ : Priority Soup Soup -> MRule .
op _->_ : Soup Soup -> MRule .
```

where a sort for labels is denoted by `Priority` because the labels are used for defining priorities between rules. We add a supersort of `MRule`, namely `MRuleSet`, used for giving the evolution rules associated to a membrane, with a concatenation operation `_ , _` which is associative, commutative, and has an identity element `none`. For example, a set of rules is given through the operator `rules` as follows:

```
eq rules(M3) = (r1 : a -> (a b, here)) ,
               (r2 : a -> (b, here) delta) ,
               (r3 : f -> (f f, here)) .
```

We define three operators: `prty`, `lhs`, and `rhs` in order to extract the components of a rule description. The partial order relation over rule priorities in a membrane is given by the operator `rho` through set of pairs of related priorities. For example, the following equation describes the priority relation in a membrane identified by a label `M1`:

```
eq rho(M1) = (r1 > r3) , (r2 > r3) .
```

We make intensive use of the order sorted equation logic implemented by Maude. Therefore we consider sort `Soup` (for multisets of objects) to be a subsort of `DissSoup` (for multisets of objects and δ's) and of `OutSoup` (for multisets of objects and out messages); `DissSoup` and `OutSoup` are subsorts of `OutDissSoup`, which in turn is a subsort of `MsgDissSoup` (for multisets of objects, messages and δ's). All these sorts are subsorts of sort `Soup?`. The sorts of the corresponding membranes are as follows: `DMembrane` for membranes having multisets of sort `DissSoup`, `Membrane+` for membranes having multisets of sort `MsgDissSoup`, and `Membrane?` for membranes having multisets of sort `Soup?`. The names of sorts for sets of membranes are composed from the name of the corresponding membrane and the suffix `Set`: `DMembraneSet`, `MembraneSet+`, and `MembraneSet?`. The sorts of nonempty sets of membranes are `NeDMembraneSet`, `NeMembraneSet+`, and `NeMembraneSet?`. We also use a sort `Configuration?` for intermediate configurations.

4.2. Maude evolution rules

As we already emphasized, a specific feature of a P system is that it has a tree like structure with the skin as its root, the composite membranes as its internal nodes and the elementary membranes as its leaves. The order of the children of a node is not important due to the associativity and commutativity properties of the concatenation operation for membranes `_ , _`.

Since Maude is not able to execute parallel transitions required by a P system, we should use a sequential application of the rules. In this sequential process, in order to prevent the case when the result of one rule is used by another, we use a technique of marking the intermediate configurations with colours. This is achieved by using two operators *blue* and *green*:

```
op blue : Membrane -> Membrane? .
op blue : MembraneSet -> MembraneSet? .
op blue : Label Soup Bool -> Soup? .
op blue : Label MRuleSet Soup Bool -> Soup?.
```

```
op green : Membrane+ -> Membrane? .
op green : MembraneSet+ -> MembraneSet? .
op green : MsgDissSoup -> Soup? .
```

The following rule marks the commitment of the maximal parallel rewriting step:

```
rl [1] : { M } => { blue(M) } .
```

starting with a term `{ M }` of sort `Configuration`.

The operator `blue` traverses the tree in a top-down manner, firing the maximal parallel rewriting process in every membrane through the `blue` operator on `Soup` terms. The Boolean argument of operation `blue` is used to show whether a dissolving rule is chosen during the current maximal parallel step in a membrane with dissolving rules. We need this flag because at most one δ symbol is allowed in a membrane. We use the variables `L:Label`, `S,S1,S2:Soup`, `PS:PrioritySet`, `neM,neM1,neM2:NeMembraneSet`, `R:MRule`, `RS:MRuleSet`. Here are the rules implementing the top-down traversal:

```
crl [3] : blue(< L | S ; neM >) => if PS == noprty
        then
             (if (freeDeltaRules(L) == none)
             then < L | blue(L, S) ; blue(neM) >
             else < L | blue(L, S, false) ; blue(neM) >
             fi)
        else
            (if (freeDeltaRules(L) == none)
            then < L | blue(L, PS, S) ; blue(neM) >
            else < L | blue(L, PS, S, false) ; blue(neM) >
            fi)
        fi
  if PS := priorities(rules(L)) .

crl [4] : blue(< L | S >) => if PS == noprty
        then
             (if (freeDeltaRules(L) == none)
             then < L | blue(L, S) >
             else < L | blue(L, S, false) >
             fi)
        else
             (if (freeDeltaRules(L) == none)
             then < L | blue(L, PS, S) >
             else < L | blue(L, PS, S, false) >
             fi)
        fi
    if PS := priorities(rules(L)) .
crl [5] : blue(neM) => blue(neM1), blue(neM2)
  if neM1, neM2 := neM .
```

where the operator `freeDeltaRules` provides the non-dissolving evolution rules of a membrane.

The multiset of objects is divided into two parts during the maximal parallel rewriting process: *blue* represents the objects available to be "consumed" via evolution rules, while the *green* represents the objects resulted from applying evolution rules over the blue objects (therefore the green objects are not available anymore for the current evolution step). When no more rule can be applied, the remaining blue objects (if any) become green.

```
crl [6] : blue(L, S) =>
            green(rhs(R)) blue(L, S1)
```

```
      if R, RS := rules(L) /\ S2 := lhs(R) /\ S2 S1 := S .
  crl [7] : blue(L, S) => green(S)
      if irreducible(L, S, rules(L)) .
```

The predicate `irreducible` returns true iff the left-hand side of each evolution rule from the membrane with the label `L` does not match the given multiset of objects. The nondeterministic choice of the evolution rules is given by the associative and commutative matching operator `R, RS := rules(L)`, and the nondeterministic choice of the objects is given by the associative and commutative matching operators `S2 := lhs(R)` and `S2 S1 := S`.

We recall that, for evolution rules with priorities, if a rule with a higher priority is used, then no rule of a lower priority can be used, even if the two rules do not compete for the same objects. In order to handle the priorities, we first introduce two new operators:

- `rmLower` — given a rule `R` and a set of rules `RS` of a membrane, this operator removes from `RS` the rules with lower priorities than the priority of `R`;
- `allowed` — a matching rule can be applied if either it does not have a priority, or it has a maximal priority, or it has a priority and any rule with a higher priority does not match the current multiset of objects.

```
crl [8] : blue(L, PS, S) =>
            green(rhs(R)) blue(L, rmLower(L, prty(R), PS), S1)
    if R, RS1 := rules(L) /\ S2 := lhs(R)
       /\ S2 S1 := S  /\ allowed(L, S, R, RS1, PS) .
crl [9] : blue(L, PS, S) => green(S)
    if irreducible(L, PS, S, rules(L)) .
```

The operator `green` traverses the tree in a bottom-up manner as follows:

– green multisets from $(O \cup Msg(O) \cup \{\delta\})^*_C$ merge into one green multiset;
– a leaf becomes green if its multiset is green;
– a set of sibling subtrees (with the roots sibling nodes) becomes green if each subtree is green;
– a subtree becomes entirely green if:

(1) the multiset of the root is green;
(2) the subtrees determined by the children of the root form a green set of sibling subtrees;
(3) there are no messages to be exchanged between the root node and its children.

We present here the rules implementing the bottom-up traversal. Note that rule 13 pushes up the operator *green* only if the corresponding membranes have no messages but may contain the dissolving symbol, i.e. the least sort of the parameter is `NeDMembraneSet`. The other rules pop up the operator *green* either from a multiset of objects, or from a multiset of membranes. We use the following additional variables: `Smd,Smd1,Smd2:MsgDissSoup`, `Sd:DissSoup`, `Sod:OutDissSoup`, `So:OutSoup`, `S':Soup?`, `neM':NeMembraneSet?`, `neM+`, `neM+,neM1+,neM2+:NeMembraneSet+`, `M1+,M2+:MembraneSet+`, `neDM,neDM1`, `neDM2:NeDMembraneSet`.

```
rl [10] : < L | green(Smd) >, neM' => green(< L | Smd >), neM' .
rl [11] : < L1 | S' ; < L | green(Smd) > > =>
          < L1 | S' ; green(< L | Smd >) > .
rl [12] : { < L | green(S) > } => { green(< L | S >) } .
rl [13] : < L | green(Sod) ; green(neDM1) > =>
          green(< L | Sod ; neDM1 >) .
rl [14] : green(Smd) green(Smd1) => green(Smd Smd1) .
rl [15] : green(neM1+), green(neM2+) => green((neM1+,neM2+))    .
```

In the communication stage of a green subtree, a node can send a message only to its parent or to one of its children. The rules for each direction of communication (*out* and *in*) vary on the structure of the destination membrane.

The rules for *in* messages are the following:

```
crl [16] : < L | green(Smd) ; green(neM+) > =>
           < L | green(Smd2) ; green(< L1 | Smd1 S >, M2+) >
           if Smd2 (S, in(L1)) := Smd /\
              < L1 | Smd1 >, M2+ := neM+ .
crl [17] : < L | green(Smd) ; green(neM+) > =>
           < L | green(Smd2) ; green(< L1 | Smd1 S ; neM1+ >, M2+) >
           if Smd2 (S, in(L1)) := Smd /\
              < L1 | Smd1 ; neM1+ >, M2+ := neM+ .
```

An object sent out of the skin membrane is lost. We allow this operation to be executed only if the skin membrane and its internal membranes are green; in this way we follow the same communication policy as for the rest of the membranes. Therefore for *out* messages we have two additional rules (20 and 21) for sending messages out of the skin membrane:

```
crl [18] : < L | green(Smd) ; green(neM+) > =>
           < L | green(Smd S) ; green(< L1 | Smd1 >, M2+) >
  if < L1 | Smd1 (S, out) >, M2+ := neM+ .
crl [19] : < L | green(Smd) ; green(neM+) > =>
           < L | green(Smd S) ; green(< L1 | Smd1 ; neM1+ >, M2+) >
  if < L1 | Smd1 (S, out) ; neM1+ >, M2+ := neM+ .
rl [20] : {< L | green(So (S, out)) >} => {< L | green(So) >} .
rl [21] : {< L | green(Smd (S, out)) ; green(neM+) >} =>
          {< L | green(Smd) ; green(neM+) >} .
```

In a P system the dissolving process occurs after the end of the communication process. To fulfil this condition we allow dissolving *only if* there are no messages to be sent (we use only variables of sort `DissSoup` and `DMembrane`). By dissolving the membrane of a node, all of its objects are transferred to the membrane of its parent, the rules are lost, and if it is an internal node, all of its children become children of its parent. The skin membrane is not allowed to be dissolved. The rules for dissolving are:

```
crl [22] : < L | Sd ; neDM > => < L | Sd S1 ; neDM1 >
           if < L1 | S1 delta >, neDM1 := neDM .
crl [23] : < L | Sd ; neDM > => < L | Sd S1 >
           if < L1 | S1 delta > := neDM .
crl [24] : < L | Sd ; neDM > => < L | Sd S ; neDM1, DM2 >
           if < L1 | S1 delta ; neDM1 >, DM2 := neDM .
```

When the entire tree becomes green, and it does not contain δ objects, there are no more messages to sent or nodes to be dissolved, the following accomplishing rule is applied:

```
rl [2] : { green(M) } => { M } .
```

The resulting term corresponds to a configuration of P system reachable in one transition step from the given configuration.

5. Operational correspondence

In this section we show how the dynamics of P systems and their corresponding implementations into Maude are related. Such a relationship between operational semantics for P systems and the Maude rewriting relation is given by two operational correspondence results.

Let $\Pi = (O, \mu, w_1, \ldots, w_n, (R_1, \rho_1), \ldots, (R_n, \rho_n), i_0)$ be a P system having the initial configuration $\langle L_1 \mid w_1 ; M_{i_1}, \ldots, M_{i_n} \rangle$, $\{i_1, \ldots, i_n\} \subseteq \{2, \ldots, n\}$, with $\mathsf{rules}(L_j) = R_j$, $\mathsf{priority}(L_j) = \rho_j$ for all membrane labels L_j, and let $\Rightarrow$ be the transition relation between two configurations. We associate to Π a rewriting theory $R(\Pi) = (\Sigma, E, R)$ in the way we presented in the previous section. Σ is the equational signature defining sorts and operation symbols, E is the set of Σ-equations which includes also the appropriate axioms for the associativity,

commutativity and identity attributes of the operators, and R is the set of rewriting rules. Considering $\longrightarrow_{R(\Pi)}$ the rewriting relation, we denote by $\longrightarrow^{+}_{R(\Pi)}$ the transitive closure of $\longrightarrow_{R(\Pi)}$, and by $\longrightarrow^{*}_{R(\Pi)}$ its reflexive and transitive closure.

An encoding function $\mathcal{I}_m : \mathcal{M}(\Pi) \to (T_{\Sigma,E})_{\texttt{Membrane}}$ from the set of membranes $\mathcal{M}(\Pi)$ to the ground terms of sort `Membrane` from the associated rewriting theory is defined by:

- if $M = \langle\ L \mid w\ \rangle$, then $\mathcal{I}_m(M) =$ `< L | w >`
- if $M = \langle\ L \mid w\ ;\ M_1, \ldots, M_n\ \rangle$,
  then $\mathcal{I}_m(M) =$ `< L | w ;` $\mathcal{I}_m(M_1)$ `,` $\ldots$ `,` $\mathcal{I}_m(M_n)$ `>`

where `L` is a constant of sort `Label`, and `w` is a term of sort `Soup`. We extend the encoding function $\mathcal{I}_m$ over nonempty sets of sibling membranes by

$$\mathcal{I}_m(M_1, \ldots, M_k) = \mathcal{I}_m(M_1), \ldots, \mathcal{I}_m(M_k),\ k \geq 2$$

We also define an encoding function $\mathcal{I} : \mathcal{C}(\Pi) \to (T_{\Sigma,E})_{\texttt{Configuration}}$ from the set of configurations to the ground terms of sort `Configuration` such that $\mathcal{I}(C) = \{\mathcal{I}_m(C)\}$, for every $C \in \mathcal{C}(\Pi)$.

We denote by $[i{-}j]$ the rewriting system consisting of the rules $[i], [i+1], \ldots, [j]$. If R and R' are two terminating rewriting systems, then t'' is a $R; R'$-normal form of t iff there is t' such that t' is a R-normal form of t and t'' is a R'-normal form of t'. $R; R'$ is a rewriting system with priorities: the rules of R have a higher priority than the rules of R'. This priority relation is different from the priority relation used in P systems. It can be implemented in the algebraic specifications of the OBJ family (including Maude) by means of sorts and the subsort relation [4].

Lemma 1. *The rewriting system consisting of the rules* [3–9] *is terminating whenever it is applied on a term of the form blue*$(M : \mathit{Membrane})$.

Proof. We define a termination function τ as follows:

$$\begin{aligned}
&\tau(\mathit{blue}(\langle\ L \mid w\ \rangle)) = 1 + |w|\\
&\tau(\mathit{blue}(\langle\ L \mid w\ ;\ t_1, \ldots, t_n\ \rangle)) = 1 + |w| + \tau(t_1, \ldots, t_n)\\
&\tau(\langle\ L \mid t\ \rangle) = \tau(t)\\
&\tau(\langle\ L \mid t\ ;\ t'\ \rangle) = \tau(t) + \tau(t')\\
&\tau(t_1, \ldots, t_n) = \tau(t_1) + \cdots + \tau(t_n)\\
&\tau(t\ t') = \tau(t) + \tau(t')\\
&\tau(\mathit{green}(w : \texttt{MsgDissSoup})) = 0\\
&\tau(\mathit{blue}(w : \texttt{Soup})) = |w|
\end{aligned}$$

For each rule of both [3–4] and [6–9], we have $\tau(lhs) > \tau(rhs)$. For rule [5], we have $\tau(lhs) = \tau(rhs)$. It is worth to note that if t' is the normal form of a term $\mathit{blue}(t : \mathit{Membrane})$ in the rewriting system [3–9], then $\tau(t') = 0$. □

Lemma 2. *The rewriting system consisting of the rules* [10–21] *is terminating whenever it is applied to a* [3–9]*-normal form.*

Proof. Let t be a [3–9]-normal form. If $\mathit{green}(t') = t|_\omega$, where ω is the sequence of positive integers describing the path from the root of t to the root of the subterm $\mathit{green}(t')$ at that position, we define $\mathit{weight}(\mathit{green}(t')) = |\omega|$. Let us define the termination function $\tau(t)$ as a pair of positive integers, where the first component is $\Sigma\{\mathit{weight}(\mathit{green}(t')) \mid \mathit{green}(t')$ subterm of $t\}$, and the second component is the number of messages in t. For each rule in [10–21], we have $\tau(lhs) >_{lex} \tau(rhs)$. □

Lemma 3. *The rewriting system consisting of the rules* [22–24] *is terminating whenever it is applied to a* [3–9]; [10–21]*-normal form.*

Proof. Let t be a [3–9]; [10–21]-normal form. We define $\tau(t)$ as being the number of δ symbols in t. We have $\tau(lhs) > \tau(rhs)$ for each rule in [22–24]. □

We assume the existence of two subsorts for membranes, namely `MprNormalForm` and `TarNormalForm`. A term t belongs to the sort `MprNormalForm` iff it is irreducible with respect to the rewriting system [3–9], and t belongs to the sort `TarNormalForm` iff it is irreducible with respect to the rewriting system [10–21]. Let $delGreen(t)$ denote the term obtained from t by deleting all occurrences of *green*, i.e. if $t = f(t_1, \ldots, t_n)$, $f \neq green$, then $delGreen(green(t)) = delGreen(t) = f(delGreen(t_1), \ldots, delGreen(t_n))$.

Proposition 5. (a) *If* $blue(L, S) \xrightarrow{*}_{[3–9]} green(S')$, *then* $S \overset{mpr}{\Longrightarrow}_L S'$.
(b) *Conversely, if* $S \overset{mpr}{\Longrightarrow}_L S'$, *then there is a rewrite* $blue(L, S) \xrightarrow{*}_{[3–9]} green(S')$.

Proof. (a) is proved by induction on the length of the rewriting, and (b) by induction on the number of rules applied in parallel. □

Proposition 6. (a) *If* t *is a* [3–9]*-normal form of* $blue(\mathcal{I}(C))$, *then there is* C' *such that* $delGreen(t) = \mathcal{I}(C')$ *and* $C \overset{mpr}{\Longrightarrow} C'$.
(b) *Conversely, if* $C \overset{mpr}{\Longrightarrow} C'$, *then* $\mathcal{I}(C') = delGreen(t)$ *for certain* t, *and* t *is a* [3–9]*-normal form of* $blue(\mathcal{I}(C))$.

Proof. (a) is proved by structural induction on C, and (b) is proved by induction on the depth of the deduction tree for $C \overset{mpr}{\Longrightarrow} C'$. □

Proposition 7. *Let* t *be a* [3–9]*-normal form of* $blue(\mathcal{I}(C))$ *such that* t *has messages and* $delGreen(t) = \mathcal{I}(C')$.
(a) *If* $green(t')$ *is a* [10–21]*-normal form of* t, *then there is* C'' *such that* $t' = \mathcal{I}(C'')$ *and* $C' \overset{tar}{\Longrightarrow} C''$.
(b) *Conversely, if* $C' \overset{tar}{\Longrightarrow} C''$, *then* $green(\mathcal{I}(C''))$ *is a* [10–21]*-normal form of* t.

Proof. (a) is proved by structural induction on C', and (b) is proved by induction on the depth of the deduction tree for $C' \overset{mpr}{\Longrightarrow} C''$. □

The proof of the next proposition is similar to the previous ones:

Proposition 8. *Let* t *be a* [3–9]; [10–21]*-normal form of* $blue(\mathcal{I}(C))$ *such that* δ *occurs in* t *and* $delGreen(t) = (C')$.
(a) *If* $green(t')$ *is a* [22–24]*-normal form of* t, *then there is* C'' *such that* $t' = \mathcal{I}(C'')$ *and* $C' \overset{\delta}{\Longrightarrow} C''$.
(b) *Conversely, if* $C' \overset{\delta}{\Longrightarrow} C''$, *then* $green(\mathcal{I}(C''))$ *is a* [22–24]*-normal form of* t.

The next theorem follows from Propositions 5–8:

Theorem 1 (*Operational Correspondence I*). *The rewriting relation given by* [1]; [3–9]; [10–21]; [22–24]; [2] *is a correct and complete implementation of* $\Rightarrow$.

5.1. Faithful and accurate implementations for P systems

The implementation of the sequential composition [3–9]; [10–21]; [22–24] using a general rewrite engine like Maude requires some auxiliary operations and verification of conditions. In this subsection we provide an alternative to the above sequential composition, discussing the consequences in defining various granularities (between "big" and "small") of the operational semantics for P systems. In what follows we replace the rules 22–24 with the following ones:

```
crl [22'] : < L | green(Sd) ; green(neDM) > =>
            < L | green(Sd S1) ; green(neDM1) >
            if < L1 | S1 delta >, neDM1 := neDM .
crl [23'] : < L | green(Sd) ; green(neDM) > =>
            < L | green(Sd S1) >
            if < L1 | S1 delta > := neDM .
crl [24'] : < L | green(Sd) ; green(neDM) > =>
            < L | green(Sd S1) ; green((neDM1, DM2)) >
            if < L1 | S1 delta ; neDM1 >, DM2 := neDM .
```

and the rule 13 with the following one:

```
rl [13'] : < L | green(Sod) ; green(neM) > =>
           green(< L | Sod ; neM >) .
```

Note that the operator *green* has a parameter *neM* of sort `NeMembraneSet` in rule 13', i.e. nonempty sets of non-dissolving membranes. In this way, the above rule together with rule 15 ensure that a subtree becomes entirely green if it is also the case that the multisets corresponding to the children and to the root do not contain δ.

We denote by RELAX the rewriting system obtained from [3–24] replacing 13 by 13', and [22–24] by [22′–24′], i.e., RELAX = [3–24]{13′/13, 22′-24′/22-24}. Experimenting in Maude by using the new implementation of the SOS of P systems, we have noticed that we are still able to simulate $\Rightarrow$ using the rewrite system [1]; RELAX; [2]. Investigating the properties of RELAX, we obtain the following results:

Proposition 9. (a) *If* $M_+ \stackrel{mpr}{\Longrightarrow}; \stackrel{tar}{\Longrightarrow} M'_+$ *and* M'_+ *is* δ*-irreducible, then green*$(\mathcal{I}_m(M'_+))$ *is a* [3–21]{13′/13}*-normal form of blue*$(\mathcal{I}_m(M_+))$.
(b) *Conversely, if green*(t') *is a* [3–21]{13′/13}*-normal form of blue*$(\mathcal{I}_m(M_+))$ *obtained by using at least one of the rules of* [16–21]*, and* t' *does not contain* δ *as subterm, then there is* M'_+ *such that* $\mathcal{I}_m(M'_+) = t'$ *and* $M_+ \stackrel{mpr}{\Longrightarrow}; \stackrel{tar}{\Longrightarrow} M'_+$.

Proposition 10. (a) *If* $M_+ \stackrel{mpr}{\Longrightarrow}; \stackrel{\delta}{\Longrightarrow} M'_+$*, then green*$(\mathcal{I}_m(M'_+))$ *is a* [3–15]{13′/13} ∪ [22′–24′]*-normal form of blue*$(\mathcal{I}_m(M_+))$.
(b) *Conversely, if green*(t') *is a* [3–15]{13′/13} ∪ [22′–24′]*-normal form of blue*$(\mathcal{I}_m(M_+))$ *obtained by using at least one of the rules of* [22′–24′]*, and* t' *does not contain messages, then there is* M'_+ *such that* $\mathcal{I}_m(M'_+) = t'$ *and* $M_+ \stackrel{mpr}{\Longrightarrow}; \stackrel{\delta}{\Longrightarrow} M'_+$.

Proposition 11. (a) *If* $M_+ \stackrel{mpr}{\Longrightarrow}; \stackrel{tar}{\Longrightarrow}; \stackrel{\delta}{\Longrightarrow} M'_+$ *then green*$(\mathcal{I}_m(M'_+))$ *is an* RELAX*-normal form of blue*$(\mathcal{I}_m(M_+))$.
(b) *Conversely, if green*(t') *is an* RELAX*-normal form of blue*$(\mathcal{I}_m(M_+))$*, obtained by using at least one of the rules of* [16–21] *and one of the rules of* [22′–24′]*, then there is* M'_+ *such that* $\mathcal{I}_m(M'_+) = t'$ *and* $M_+ \stackrel{mpr}{\Longrightarrow}; \stackrel{tar}{\Longrightarrow}; \stackrel{\delta}{\Longrightarrow} M'_+$.

In the new rewriting *blue*$(\mathcal{I}(C)) \stackrel{*}{\rightarrow}_{\mathsf{RELAX}}$ *green*(t') we do not have a strict separation between the internal steps given by the evolution of membranes, communication, and dissolving. If two parent-child membranes finish their internal evolution, then they can communicate without waiting for the other membranes of the system to finish their evolution step. Similarly, if two parent-child membranes finish the communication, then the child may dissolve whenever it has a δ object.

The next theorem follows from the previous results (Propositions 9–11):

Theorem 2 (*Operational Correspondence II*). *The rewriting system* RELAX *is terminating on blue*$(\mathcal{I}(C))$ *for any configuration* C*. Moreover, if* $C \Rightarrow C'$*, then green*$(\mathcal{I}(C'))$ *is an* RELAX*-normal form of blue*$(\mathcal{I}(C))$ *and, conversely, if green*(t') *is an* RELAX*-normal form of blue*$(\mathcal{I}(C))$*, then there is* C' *such that* $t' = \mathcal{I}(C')$ *and* $C \Rightarrow C'$.

Theorems 1 and 2 reveal two forms of correctness of an implementation with respect to the given operational semantics. In a stronger form, we say that an implementation $\mathcal{I}$ is *faithful* if and only if it is defined by three relations $\rightsquigarrow_{mpr}$, $\rightsquigarrow_{tar}$, and $\rightsquigarrow_{diss}$ such that $\mathcal{I}(C) \rightsquigarrow_{mpr} \mathcal{I}(C_1)$ whenever $C \stackrel{mpr}{\Longrightarrow} C_1$, $\mathcal{I}(C_1) \rightsquigarrow_{tar} \mathcal{I}(C_2)$ whenever $C_1 \stackrel{tar}{\Longrightarrow} C_2$, and $\mathcal{I}(C_2) \rightsquigarrow_{diss} \mathcal{I}(C')$ whenever $C_2 \stackrel{\delta}{\Longrightarrow} C'$, for all configurations C, C' and intermediate configurations C_1, C_2. In a weaker form, we say that an implementation $\mathcal{I}$ is *accurate* if and only if it is defined by a relation $\rightsquigarrow$ such that $\mathcal{I}(C) \rightsquigarrow \mathcal{I}(C')$ whenever $C \Rightarrow C'$, for all configurations C, C'. Theorem 1 refers to a faithful implementation, and Theorem 2 refers to an accurate implementation.

In an accurate implementation it is possible to execute parallel transitions of different phases. This fact can increase the potential parallelism of the rewriting implementations of the P systems. On the other hand, a faithful implementation generates a smaller state space.

The computation of a RELAX-normal form requires fewer auxiliary rewritings and verifying conditions than the computation of a [3–9]; [10–21]; [22–24]-normal form. However, if we use the Maude model-checker for analyzing P systems, then [3–9]; [10–21]; [22–24] could be more efficient because it generates a smaller state space. We can

exemplify this aspect by using a simple P system $\langle L_1 \mid aa; \langle L_2 \mid yy; \langle L_3 \mid vv\rangle\rangle\rangle$, where $rules(L_1) = a \to (b, in(L_2))$, $rules(L_2) = y \to (x, out)(z, in(L_3))$, and $rules(L_3) = v \to (u, out)$. We use the Maude command `search` to generate the whole state space. For the faithful implementation we get 292 states:

```
Maude> search init =>+ C:Configuration .
search in EX : init =>+ C:Configuration .

Solution 1 (state 262)
states: 263  rewrites: 3059 in 41ms cpu (42ms real)
C:Configuration --> {< L1 | x x ; < L2 | b b u u ; < L3 | z z > > >}

No more solutions.
states: 292  rewrites: 3420 in 45ms cpu (46ms real)
```

For the accurate implementation we get 752 states:

```
Maude> search init =>+ C:Configuration .
search in EX : init =>+ C:Configuration .

Solution 1 (state 731)
states: 732  rewrites: 11001 in 90ms cpu (90ms real)
C:Configuration --> {< L1 | x x ; < L2 | b b u u ; < L3 | z z > > >}

No more solutions.
states: 752  rewrites: 11371 in 92ms cpu (92ms real)
```

Since we have a higher level of parallelism in the accurate implementation, there are more paths from the initial configuration to the unique final one. The additional states are given by configurations where different membranes are in different phases, e.g. L_1 evolves, while L_2 and L_3 communicate. These forms of implementation exhibit different levels of parallelism which can be exploited in analyzing P systems. An accurate implementation can be used to analyze more aspects inspired by biology. Such an implementation is also appropriate when we are interested in speeding up the execution on a parallel machine. On the other hand, if we are interested to investigate only the configurations (states), then it is better to use a faithful implementation. We can use additionally the power of the abstraction mechanism supplied by rewriting logic. The state space can be drastically reduced if the rules describing the top-down and bottom-up traversals are transformed into equations. Making this modification in the faithful implementation, we get only 97 states:

```
Maude> search init =>+ C:Configuration .
search in EX : init =>+ C:Configuration .

Solution 1 (state 83)
states: 84  rewrites: 1846 in 17ms cpu (17ms real)
C:Configuration --> {< L1 | x x ; < L2 | b b u u ; < L3 | z z > > >}

No more solutions.
States: 97  rewrites: 2090 in 20ms cpu (20ms real)
```

This modification is sound because the rewriting systems defined by these rules are terminating and confluent. The price paid is that the properties of some intermediate states cannot be observed.

The accurate implementation leads to a new operational semantics where the phases of maximal parallel rewriting, communication and dissolving are not strongly delimited. We think that this new small-step operational semantics can provide a richer class of models described by P systems.

6. Conclusion and related work

We start by presenting some related models. Multisets represent the fundamental structure of other computation models such as Gamma and Chemical Abstract Machine. As in P systems, multisets are viewed as entities containing data (possibly, other multisets), and some multiset rules can be applied with a high degree of parallelism. The Gamma formalism was proposed in [5] to capture the intuition of computation as a global evolution of a collection of atomic values interacting freely. The computation model underlying Gamma is based on the chemical reaction metaphor; the data are considered as a multiset of molecules and the computation is described by chemical reactions according to some rules. Several reactions happen at the same time. The generality of the rules ensures a great expressive power and, in a direct manner, computational universality. Even closer to P systems are the structured multisets defined in [13]. Structured multisets can be seen as a syntactic facility allowing the organization of explicit data, and providing a notation leading to higher-level programs manipulating more complex data structures.

The CHemical Abstract Machine (CHAM) formalism [6] extends the Gamma formalism introducing the notion of sub-solution enclosed in a membrane, together with a given concurrency between the rules as a primitive notion. Membranes appear in CHAM, but they are not similar to membranes in the P systems; they correspond to the contents of membranes (multisets and lower level membranes). The goal of CHAM is completely different from that of P systems. CHAM is mainly directed to the algebraic treatment of the processes these membranes can undergo; they are more related to the recent introduced Brane calculus [8]. What Gamma and CHAM do not have is the notion of localization, and the distribution aspects described easily by P systems.

Multiset rewriting lies at the core of these formalisms. Multiset rewriting is a special case of rewriting rules where the operators considered are both associative and commutative. Several frameworks provide efficient environments to apply multiset rewriting rules, eventually following some strategies. Among them, Maude and Elan [7] are two such systems.

MGS is designed to represent and manipulate local transformations of entities structured by abstract topologies [14]. A set of entities organized by an abstract topology is called a topological collection. The collection types can range in MGS from sets and multisets to more structured types. MGS has the ability to nest different topologies in order to describe biological systems. MGS can use transformation on multisets to express the computations of Gamma and of the membrane systems. However MGS has a total order over rules.

Among these related approaches and formalisms with common ideas and differences between them, we have selected membrane computing because it is directly inspired by the cell biology, and uses new and useful ideas: localization, hierarchical structures, distribution, communication. The P systems provides an elegant and powerful computation model, able to solve computationally hard problems in a feasible time, and useful to model various biological phenomena.

In the framework of the membrane computing, we try to take advantage of good features of both structural operational semantics and rewriting logic. In this way we use structural operational semantics to provide a detailed step-by-step modelling of P systems computation. Operational semantics has a simple proof-theoretic semantics, and it allows mathematical reasoning and proofs, by reasoning inductively or co-inductively about the inference steps.

A first Maude implementation is given in [1]; then a model-checking algorithm is applied for the first time to analyze P systems. This implementation uses intensively the reflection property of rewriting logic, and is given for P systems without dissolving. Since the analysis is given at meta-metalevel, it is time consuming. Trying to extend the implementation for P systems with dissolving and other control mechanisms, we noticed that the main difficulty comes from the lack of a formal operational semantics for P systems. The existence of such a formal description allows to prove the correctness and completeness of the implementations. An operational semantics defining a deductive system is presented in [2]. Starting from this formal semantics, we define a new implementation. A first version of that implementation is presented in [3]. In this paper we extend the presentation from [3], and we show that implementing the natural semantics of P systems into rewriting logic can help us to understand better the new model provided by P systems. Moreover, rewriting logic semantics reveals the existence of various granularities in defining a small-step operational semantics of the membrane systems. We prove the correctness and completeness of the implementations with respect with the operational semantics. All these implementations can be found at http://thor.info.uaic.ro/~rewps/index.html.

The advantages of the implementations in Maude is given by the solid theoretical aspects of the rewriting logic, and by the complex tools available in Maude. Among other several implementations of the membrane systems, we

mention here only an implementation given on a parallel architecture, and few sequential implementations used in solving NP-complete problems. A parallel implementation for transition membrane systems is reported in [10]. The rules are implemented as threads. At the initialization phase, one thread is created for each rule. Rule applications are performed in terms of rounds. To synchronize each thread (rule) within the system, two barriers implemented as mutexes are associated with the thread. The implementation was designed for a cluster of computers. It is written in C++ and it makes use of Message Passing Interface (MPI) as its communication mechanism.

Transition membrane systems and deterministic membrane systems with active membranes are simulated in Prolog [12]; they are used to solve NP-complete problems as SAT, VALIDITY, Subset Sum, Knapsack, and partition problems. Membrane systems with active membranes, input membrane and external output are simulated in CLIPS; they are used to solve NP-complete problems in [18]. Polynomial-time solutions to NP-complete problems via membrane systems can be reached trading time by space. This is done by producing an exponential amount of membranes which can work in parallel.

Acknowledgement

The second and third authors' research was partially supported by Romanian CEEX Grant 47/2005.

References

[1] O. Andrei, G. Ciobanu, D. Lucanu, Executable specifications of the P systems, Lecture Notes in Computer Science 3365 (2005) 127–146.
[2] O. Andrei, G. Ciobanu, D. Lucanu, A structural operational semantics of the P systems, Lecture Notes in Computer Science 3850 (2006) 32–49.
[3] O. Andrei, G. Ciobanu, D. Lucanu, Operational semantics and rewriting logic in membrane computing, Electronic Notes of Theoretical Computer Science 156 (2006) 57–78.
[4] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, W.P. Weijland, Term-rewriting systems with rule priorities, Theoretical Computer Science 67 (1989) 283–301.
[5] J.-P. Banatre, D. Le Metayer, A new computational model and its discipline of programming, INRIA Research Report No. 566, 1986.
[6] G. Berry, G. Boudol, The chemical abstract machine, Theoretical Computer Science 96 (1992) 217–248.
[7] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, ELAN from the rewriting logic point of view, Theoretical Computer Science 285 (2002) 155–185.
[8] L. Cardelli, Brane calculi, Lecture Notes in Computer Science 3082 (2005) 257–278.
[9] G. Ciobanu, Distributed algorithms over communicating membrane systems, Biosystems 70 (2003) 123–133.
[10] G. Ciobanu, W. Guo, P systems running on a cluster of computers, Lecture Notes in Computer Science 2933 (2004) 123–139.
[11] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J. F. Quesada, Maude: Specification and programming in rewriting logic, Theoretical Computer Science 285 (2002) 1870–1243.
[12] A. Cordon-Franco, M.A. Gutierrez-Naranjo, M.J. Perez-Jimenez, A. Riscos-Nunez, F. Sancho-Caparrini, Implementing in prolog an effective cellular solution for the Knapsack problem, Lecture Notes in Computer Science 2933 (2004) 140–152.
[13] P. Fradet, D. Le Metayer, Structured gamma, Science of Computing Programming 31 (1998) 263–289.
[14] J.-L. Giavitto, O. Michel, MGS: A rule-based programming language for complex objects and collections, Electronic Notes of Theoretical Computer Science 59 (2001).
[15] J. Meseguer, G. Rosu, Rewriting logic semantics: From language specifications to formal analysis tools, Lecture Notes in Computer Science 3097 (2004) 1–44.
[16] Gh. Păun, Membrane Computing. An Introduction, Springer, 2002.
[17] Gh. Păun, Introduction to membrane computing, in: Proc. 1st Brainstorming Workshop on Uncertainty in Membrane Computing, 2004, pp. 17–65.
[18] M.J. Perez-Jimenez, F.J. Romero-Campero, A CLIPS Simulator for Recognizer P Systems with Active Membranes, University of Sevilla Techical Report 01/2004, 2004, pp. 387–413.
[19] G.D. Plotkin, Structural operational semantics, Journal of Logic and Algebraic Programming 60 (2004) 17–139.